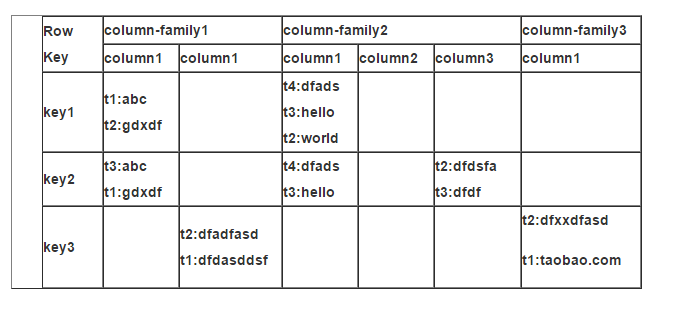
# 表结构设计

http://blog.csdn.net/codestinity/article/details/6981278

逻辑视图

HBase以表的形式存储数据。表有行和列组成。列划分为若干个列族(row family)



**Row Key**

与nosql数据库们一样,row key是用来检索记录的主键。访问hbase table中的行，只有三种方式：

1 通过单个row key访问

2 通过row key的range

3 全表扫描（scan）

Row key行键 (Row key)可以是任意字符串(最大长度是 64KB，实际应用中长度一般为 10-100bytes)，在hbase内部，row key保存为字节数组。

存储时，数据按照Row key的字典序(byte order)排序存储。设计key时，要充分排序存储这个特性，将经常一起读取的行存储放到一起。(位置相关性)

注意：

字典序对int排序的结果是1,10,100,11,12,13,14,15,16,17,18,19,2,20,21,…,9,91,92,93,94,95,96,97,98,99。要保持整形的自然序，行键必须用0作左填充。

行的一次读写是原子操作 (不论一次读写多少列)。这个设计决策能够使用户很容易的理解程序在对同一个行进行并发更新操作时的行为。

HBase不支持条件查询和Order by等查询，读取记录只能按Row key（及其range）或全表扫描，因此Row key需要根据业务来设计以利用其存储排序特性（Table按Row key字典序排序如1,10,100,11,2）提高性能。

**列族**

hbase表中的每个列，都归属与某个列族。列族是表的chema的一部分(而列不是)，必须在使用表之前定义。列名都以列族作为前缀。例如*courses:history ， courses:math 都属于 courses* 这个列族。

访问控制、磁盘和内存的使用统计都是在列族层面进行的。实际应用中，列族上的控制权限能帮助我们管理不同类型的应用：我们允许一些应用可以添加新的基本数据、一些应用可以读取基本数据并创建继承的列族、一些应用则只允许浏览数据（甚至可能因为隐私的原因不能浏览所有数据）。

**时间戳**

HBase中通过row和columns确定的为一个存贮单元称为cell。每个 cell都保存着同一份数据的多个版本。版本通过时间戳来索引。时间戳的类型是 64位整型。时间戳可以由hbase(在数据写入时自动 )赋值，此时时间戳是精确到毫秒的当前系统时间。时间戳也可以由客户显式赋值。如果应用程序要避免数据版本冲突，就必须自己生成具有唯一性的时间戳。每个 cell中，不同版本的数据按照时间倒序排序，即最新的数据排在最前面。

为了避免数据存在过多版本造成的的管理 (包括存贮和索引)负担，hbase提供了两种数据版本回收方式。一是保存数据的最后n个版本，二是保存最近一段时间内的版本（比如最近七天）。用户可以针对每个列族进行设置。

**Cell**

由*{row key, column(* =<family> + <label>*), version}* 唯一确定的单元。cell中的数据是没有类型的，全部是字节码形式存贮。

# shell常用指令

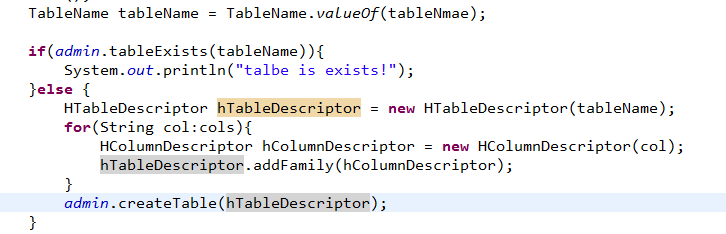
下面我们看看HBase Shell的一些基本操作命令，我列出了几个常用的HBase Shell命令，如下：



## ****DDL操作（数据定义语言）****

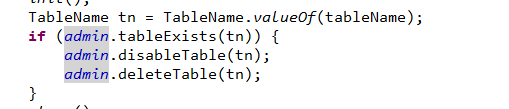
#### 建表

shell ：create '表名称', '列名称1','列名称2','列名称N'



#### 删除表

先进行disable操作，在进行drop操作



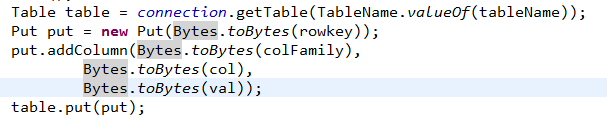
http://blog.csdn.net/scutshuxue/article/details/6988348

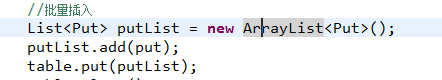
http://blog.csdn.net/rkjava/article/details/48789683

Get、Put、Delete、Scan、Increment（递增）

## DML操作（数据操控语言）

#### 增加数据





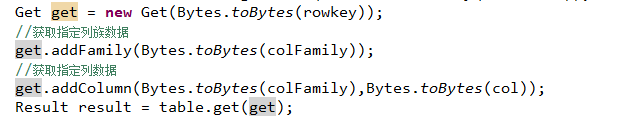
为什么不直接存储数据呢？hbase中所有的数据都是作为原始数据使用字节数组的形式存储，行键也是如此。

#### 更新数据

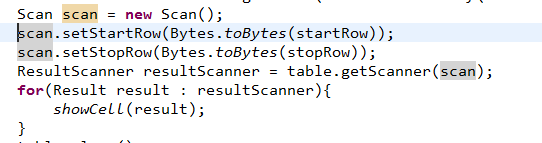
mongodb的update是update-in-place，也就是原地更新，除非原地容纳不下更新后的数据记录。而hbase的修改和添加都是同一个命令：put，如果put传入的row key已经存在就更新原记录,实际上hbase内部也不是更新，它只是将这一份数据已不同的版本保存下来而已，hbase默认的保存版本的历史数量是3。

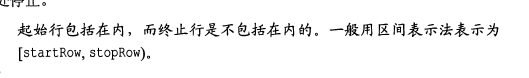
#### 获取数据

1 通过单个row key访问

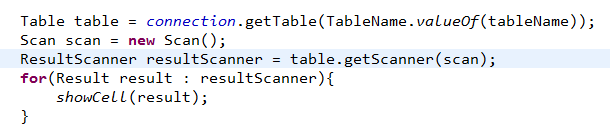


2 通过row key的range





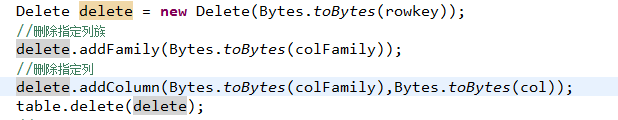
3 全表扫描（scan）

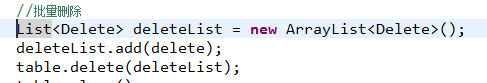


#### 删除数据

Hbase的delete是先新建一个tombstonemarkers，然后读的时候会和tombstonemarkers做merge，在 发生major compaction时delete的数据记录才会真真删除。

HFile 是不可改变的，当我们发送一条删除指令的时并不能直接在HFile中进行数据的删除，得等到合并操作才会真正删除数据，这个是设计的原因。





http://blog.csdn.net/yfkiss/article/details/26380467

## 设计

从应用角度，有两点比较重要：  
**1. HBase中RowKey是按照字典序排列的  
2. 不同Column Family的数据，在物理上是分开的**  
在做table设计的时候，主要围绕上述两点做文章。  
RowKey的设计需要根据请求数据特点：  
1.单个查询，**需要尽量缩小Key的长度**  
2.范围查询，根据RowKey按字典序排列的特点，针对查询需求设计rowkey，保证范围查询的rowkey在物理上存放在一起  
Column Family的设计需遵循：**尽量避免一次请求需要拿到的Column分布在不同的Column Family中**

## 实例

对于基于RowKey的范围查询设计，我们来看**一个实例**：  
1. 给出userid，返回这个userid最近插入的N条数据   
2. 给出userid，及一个时间区间，返回这个时间区间的N条数据  
  
针对需求，Key设计如下：  
**Userid\_DataTime\_InertTime**  
Userid：即userid  
DataTime：数据所属时间(ms)，定义为：Long.MAX\_VALUE - dataTime.getTime()，由于RowKey字典序排列，可以使最近插入的数据排在前面，支持“最近插入的N条数据”的需求  
InsertTime：数据入库时间(ns)，取nanotime（InsertTime的存在是由于在这个应用中，Userid+DataTime不能唯一定位一条数据）